

# 15-214: Principles of Software Construction

8<sup>th</sup> March 2012

Name:   
Recitation Section (or Time):

**Instructions:**

- Make sure that your exam is not missing any sheets (it should contain **13 pages**).
- Write your **full name** on this page and **Andrew ID** on the header of all.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The problems are of varying difficulty. The point value of each problem is indicated.
- Ideally you should take **one minute per point**. So pace yourself accordingly.
- This exam is **CLOSED BOOK**. You may not use a calculator, laptop or any other electronic or wireless device.
- Write **concise** and **focused** answers (long ramblings will hurt your grade).
- If anything is unclear, just make and state your (reasonable) assumptions.
- Make sure your handwriting is **READABLE**.

You have **80 minutes** to complete this Exam. Good luck!

Question	Points	Score
Java	15	
Unit Testing	10	
Method Dispatch	16	
Design Patterns	16	
Verification	22	
<b>TOTAL:</b>	79	

**Java [15 points]**

Q1. [5 points] Fill in the blanks:

- a)  is the feature that allows different implementations of the same interface to behave differently.
- b)  is the capability of a class to manifest the properties and methods of another class while adding its own functionality.
- c) Java permits a class to replace the implementation of a method that it has inherited. It is called .
- d)  keyword assigned to a class member to hide that member from all other classes.
- e) The main method is defined to be  because it is not a property of an instance but of the class.

Q2. [2 points] Print out the result:

```
String a = new String("4");
String b = new String("4");
System.out.print( a.equals(b) );
System.out.print( a == b );
```

  

Q3. [4 points] Find **two** things that cause this code not to compile? Mark the errors.

```
public final abstract class Animal{
    public void speak(){
        System.out.println("My name is " + getName() + ".");
    }
    public String getName();
}

public class Dog extends Animal{
    public String getName(){
        return "Tommy";
    }
    public static void main(String[] args){
        Animal d = new Dog();
        d.speak();
        d.getName();
    }
}
```

Q4. [4 points] Given below is a portion of the inheritance hierarchy of the various errors and exceptions. Please use this to answer the following questions.

```
class java.lang.Throwable (implements java.io.Serializable)
  class java.lang.Error
    class java.lang.AssertionError
    class java.lang.LinkageError
      class java.lang.ClassCircularityError
      class java.lang.ClassFormatError
        class java.lang.UnsupportedClassVersionError
      class java.lang.ExceptionInInitializerError
      class java.lang.IncompatibleClassChangeError
        class java.lang.AbstractMethodError
        class java.lang.IllegalAccessError
        class java.lang.InstantiationError
        class java.lang.NoSuchFieldError
        class java.lang.NoSuchMethodError
```

```
class java.lang.NoClassDefFoundError
class java.lang.UnsatisfiedLinkError
class java.lang.VerifyError
class java.lang.ThreadDeath
class java.lang.VirtualMachineError
class java.lang.InternalError
class java.lang.OutOfMemoryError
class java.lang.StackOverflowError
class java.lang.UnknownError
class java.lang.Exception
class java.lang.ClassNotFoundException
class java.lang.CloneNotSupportedException
class java.lang.IllegalAccessException
class java.lang.InstantiationException
class java.lang.InterruptedException
class java.lang.NoSuchFieldException
class java.lang.NoSuchMethodException
class java.lang.RuntimeException
class java.lang.ArithmeticException
class java.lang.ArrayStoreException
class java.lang.ClassCastException
class java.lang.IllegalArgumentException
class java.lang.IllegalThreadStateException
class java.lang.NumberFormatException
class java.lang.IllegalMonitorStateException
class java.lang.IllegalStateException
class java.lang.IndexOutOfBoundsException
class java.lang.ArrayIndexOutOfBoundsException
class java.lang.StringIndexOutOfBoundsException
class java.lang.NegativeArraySizeException
class java.lang.NullPointerException
class java.lang.SecurityException
class java.lang.UnsupportedOperationException
```

a) What does the following program print?

```
public static void main(){
    try{
        String s = "";
        Char b = s.charAt(5);
    } catch ( Exception e ){
        System.out.println("caught an exception");
    } catch ( Error e ){
        System.out.println("caught an error");
    }
}
```

b) What does the following program output?

```
public static void main(){
    try{
        assertTrue(false);
    } catch (Exception e){
        System.out.println("caught an exception");
    } catch (Error e){
        System.out.println("caught an error");
    }
}
```

## Unit Testing [10 points]

You are to develop a **set of unit test cases** for the following code block.

Note that the code below may contain some faults. The test cases you write should catch these faults and any others that might arise as the code evolves.

```
public class AccumValue {

    private int value;

    public AccumValue(int init){ value = init; }

    /**
     * Computes a new accumulation value from the values in vs up to limit
     * @returns true if value changed, false otherwise.
     */
    public boolean calcAccum ( int limit, AccumValue[] vs ){
        int[] array = new int[limit];

        if( array.length < limit )
            return false;

        buildArray(array,vs);

        int tmp=0;
        for( int s : array )
            tmp += s;

        int old = value;
        value = tmp;
        return old != value;
    }

    protected void buildArray( int[] array, AccumValue[] vs ){
        for( int i=0 ; i<array.length ; ++i )
            array[i] = vs[i].getValue();
    }

    public int getValue(){ return value; }
}
```

Q1. [10 points] Fill in the blanks to produce several test cases that check the code shown above.

If a test case should throw an exception you should write it as:

```
@Test( expected = NameOfException.class )
```

where NameOfException is a reasonable exception that the code should throw for that situation. If the test case should not throw any exception then either cross out the blank box or leave it empty.

```
public class AccumValueTester {

    @Test 
    public void testConstructor() {
        assertEquals( 214 , new AccumValue(214).  );
    }

    @Test 
    public void testArgumentsSanityCheck1() {
        new AccumValue(0).calcAccum(  ,  );
    }

    @Test 
    public void testArgumentsSanityCheck2() {
        new AccumValue(0).calcAccum(  ,  );
    }

    @Test 
    public void testArgumentsSanityCheck3() {
        new AccumValue(0).calcAccum(  ,  );
    }

    @Test 
    public void testCorrectResult() {
        AccumValue c = new AccumValue(0);
        AccumValue[] x = { new AccumValue(2), new AccumValue(3) };
        assertTrue(  );
        assertEquals(  ,  );
        assertFalse(  );
        assertEquals(  ,  );
        assertTrue(  );
        assertEquals( 5 , c.getValue() );
    }

}
```

## Method Dispatch [16 Points]

Consider the following code:

```
package b;
abstract class Bird {
    protected int a = 1;
    public abstract void identify();
    protected void shout(){
        System.out.println("Kaw-Kaw");
    }
    public void action(){
        System.out.println("Fear me "+ this.a +" times");
    }
    public void flyAround(){
        System.out.println("Woooooosh, I am a ");
        identify();
        shout();
        System.out.println("You should ");
        action();
    }
    protected int getA(){ return a; }
    protected void addToA(){ a++; }
}

package b;
public class Penguin extends Bird{
    private int a;
    public Penguin(){ a = 2; }
    @Override
    public void identify() {
        System.out.println("Penguin #" + a + " reporting for duty!");
    }
    public void shout(){
        System.out.println("Shouting is uncivilized...");
    }
    public void action(int times){
        System.out.println("Swim "+ (times + getA()) +" times!");
    }
    public void flyAround(){
        System.out.println("Sometimes I dream I can fly like this: ");
        super.flyAround();
    }
}

package a;
public class Program{
    public static void act1(){
        Bird b = new Bird();
        b.flyAround();
    }
}
```

```
public static void act2(){
    Bird b = new Penguin();
    act3(b);
}
public static void act3(Penguin p){
    Bird b = new Penguin();
    b.shout();
}
public static void act4(){
    Bird b = new Penguin();
    b.action(15);
}
public static void act5(){
    Bird b = new Penguin();
    b.flyAround();
}
public static void act6(){
    Bird b = new Penguin();
    b.identify();
}
public static void act7(){
    Bird b = new Penguin();
    b.addToA();
    ((Penguin)b).action(4);
}
}
```

Q1. [16 points] For each of the following methods that refer to the code above, if there is a compilation error, explain what it is and how to fix it. If there is no compilation error in a particular act method, say what the method prints.

a) Program.act1()

b) Program.act2()

c) Program.act3(new Penguin())

d) Program.act4()

e) Program.act5()

f) Program.act6()

g) Program.act7()



## Design Patterns [16 points]

```
public interface Dead {
    public Beef dead1();
}
public interface Beef {
    public void beef1(Foo f);
    public void beef2();
}
public interface Foo {
    public void foo1();
}
public class Deadbeef implements Dead {
    private static Deadbeef feebdaed = new Deadbeef();
    private Deadbeef(){
    }
    private static DeadBeef deadbeef1(){
        return feebdaed;
    }
    public Beef dead1(){
        return new Beefdead();
    }
}
public class Beefdead implements Beef {
    private ArrayList<Foo> beefdead = new ArrayList<Foo>();
    public void beef1(Foo f){
        beefdead.add(f);
    }
    public void beef2(){
        for(Foo f: beefdead){
            f.foo1();
        }
    }
}
public class Bar implements Foo {
    public void foo1(){
        System.out.println("15-214 rocks my socks!");
    }
}
```

The world-renowned hacker, "NotYour214TA", has some tricks up his sleeve. In order to protect his code from being understood by other he changes around all of the names so that other hackers can't read it. Little does he know at Carnegie Mellon, we laugh at such simple problems!

Q1. [12 points] There are 3 key design patterns that are used in the code above. State these patterns and describe which classes play which roles in those patterns.

*Hint 1: CHOOSE FROM THESE PATTERNS:* singleton, facade, adapter, strategy, proxy, composite, observer, factory method, template, decorator.

**PATTERNS THAT ARE NOT IN THIS LIST WILL NOT RECIEVE CREDIT**

*Hint 2:* A single class may be used in multiple patterns.

**pattern #1:**

**role of each relevant class in pattern #1:**

**pattern #2:**

**role of each relevant class in pattern #2:**

**pattern #3:**

**role of each relevant class in pattern #3:**

Q2 . [4 points] Name the most appropriate pattern for each purpose:

*Hint:* **CHOOSE FROM THESE PATTERNS:** singleton, facade, adapter, strategy, proxy, composite, observer, factory method, template method, decorator.

- a) add functionality to individual objects dynamically and transparently
  
- b) expose the functionality of an object through another interface
  
- c) fix the structure of an algorithm but allow portions of that algorithm to vary
  
- d) load an object from a database on demand

**Verification [22 points]**

Q1. [8 points] Compute the weakest precondition.

Assume integer operations. Simplify the result as much as possible.

*Important note: Assume all variables and operations range over integers.*

a)  $\text{wp}('r *= x; n = n-1', \text{ans} = x^n * r)$

b)  $\text{wp}('x *= x; n /= 2', \text{ans} = x^n * r)$

Q2. [6 points] Always (all models), Never (no models), or Sometimes (some models)

a) True $\Rightarrow$ False	<input type="text"/>
b) False $\Rightarrow$ False	<input type="text"/>
c) False $\Rightarrow$ True	<input type="text"/>
d) False $\Rightarrow$ $x=1$	<input type="text"/>
e) $x=1 \Rightarrow$ False	<input type="text"/>
f) $x=1 \Rightarrow$ True	<input type="text"/>

Q3. [8 points] The weak and the strong. Indicate your answer by circling an assertion or by notating "None of these."

a) Which assertion is **weakest**?

$x=1$   
Odd( $x$ )  
True  
*None of these*

b) Which assertion is **strongest**?

$x=1$   
Odd( $x$ )  
True  
*None of these*

c) Which assertion is **strongest**?

$x=1$   
Odd( $x$ )  
False  
 $y=1$   
*None of these*

d) Which assertion is **weakest**?

$x=1$   
Odd( $x$ )  
False  
 $y=1$   
*None of these*